Optimizing Fully Anisotropic Elastic Propagation on 2nd Generation Intel Xeon Phi Processors

1 Introduction

In the last few years, acoustic isotropic wave propagation has been the preferred simulation engine for 3D full-wave field modelling-based applications. The reason for its success over better approximations is the lower computational cost it entails. However, recent trends in seismic imaging rely on an improved physical model that represents the Earth no more as a rigid body but as an elastic and anisotropic one. Also, moving the wave propagation simulation closer to the real physics of the problem results in a significant increment of the needed computational resources.

New hardware alternatives appear as a potential solution to satisfy the high demands of the computing power of the elastic, anisotropic, wave propagation engine. Also, the last decade has seen a trend on building systems with dedicated devices and accelerators, which produce a good FLOPs/Watt ratio. One of the most promising HPC alternatives comes from Intel®Xeon Phi™ product family.

This work shows several optimization strategies evaluated and applied to an elastic wave propagation engine, based on a Fully Staggered Grid, running on the latest Intel Xeon Phi processors, the second generation of the product (code-named Knights Landing). The developed propagator is able to reproduce elastic wave propagation, even for an arbitrary anisotropy.

2 Wave propagation in anisotropic elastic media

Under small deformations, propagation of seismic waves can be modeled by Newton's second law and the linearly elastic stress-strain relationships [4]. Surface forces on an elastic body comprises six different stress components, three compressional stresses σ_{xx}, σ_{yy} and σ_{zz}, and three shear stresses σ_{yz}, σ_{xz} and σ_{xy}, where subscripts stand for the face and direction of force application. Stresses induce material strains that are quantified in terms of displacement gradients. Time differentiation of the stress-strain relations (Eqs.1) coupled to the Newton's equation of motion (2) allows describing wave propagation in an elastic medium. In this system, u, v, and w represent the particle velocity components in the $x-$, $y-$ and $z-$ directions, respectively, and the stiffness tensor C defines the anisotropic response of the material.

$$\begin{pmatrix} \partial_t \sigma_{xx} \\ \partial_t \sigma_{yy} \\ \partial_t \sigma_{zz} \\ \partial_t \sigma_{yz} \\ \partial_t \sigma_{xz} \\ \partial_t \sigma_{xy} \end{pmatrix} = C \begin{pmatrix} \partial_x u \\ \partial_y v \\ \partial_z w \\ \partial_z v + \partial_y w \\ \partial_z u + \partial_x w \\ \partial_x v + \partial_y u \end{pmatrix} \tag{1}$$

$$\begin{aligned} \rho \partial_t u &= \partial_x \sigma_{xx} + \partial_y \sigma_{xy} + \partial_z \sigma_{xz} \\ \rho \partial_t v &= \partial_x \sigma_{xy} + \partial_y \sigma_{yy} + \partial_z \sigma_{yz} \\ \rho \partial_t w &= \partial_x \sigma_{xz} + \partial_y \sigma_{yz} + \partial_z \sigma_{zz} \end{aligned} \tag{2}$$

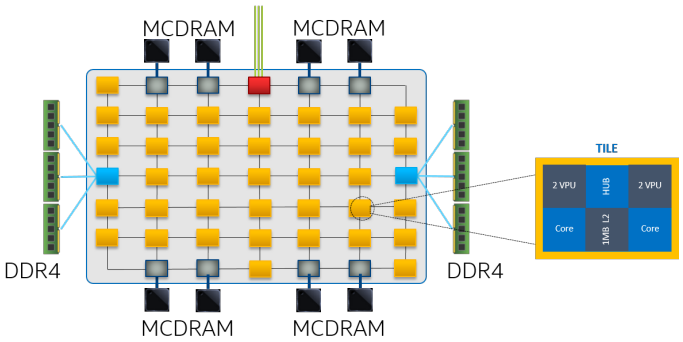

Figure 1: Intel Xeon Phi x200 processor family architecture overview.

The tensor C is always a symmetric matrix with 21 arbitrary components in the most general anisotropic case. However, certain materials can be described with fewer parameters, and the most frequent symmetry types are monoclinic, orthorhombic and transversely isotropic, with 13, 9 and 5 independent parameters, respectively. On the other end, only the two Lamé constants permit defining the non zero C entries that model fully isotropy. In this work, we will focus on the complete C matrix to allow arbitrary anisotropy and nearly realistic topography [3].

3 The Intel Xeon Phi x200 processor family

For this study we are using a system with a CPU from the Intel Xeon Phi x200 processor family (code-named Knights Landing). As shown in Figure 1 these processors have a multi-core architecture with up to 36 tiles per package connected through a 2D mesh between them and the memory controllers. Each tile is composed of two cores that share a 512 MB L2 cache and an agent that connects the tile to the mesh. Each core appears as four logical CPUs through hyper-threading. All hyper-threads on a core share a first-level (L1) cache. The cores are capable of issuing two instructions per cycle out-of-order, including vector and memory instructions. The Intel Xeon Phi architecture implements the Intel®Advanced Vector Extensions 512 (Intel®AVX-512) instruction set. This instruction set provides 512-bit SIMD instructions where each SIMD register may contain eight double-precision (DP) or sixteen single-precision (SP) floating-point values as well as a variety of integer data sizes. The instruction set also supports low-overhead unaligned loads, fused-multiply and add, vector masking, shuffle and permutation instructions, advanced vector gather and scatter, histogram support, and hardware-accelerated transcendentals. In addition, the Intel Xeon Phi processor also supports other instructions sets supported by Intel®Xeon®processors.

To be able to provide the cores with enough data to feed the computing

capabilities, the Intel Xeon Phi processors may be configured with an integrated on-package Multi-Channel DRAM (MCDRAM) memory of up to 16 GiB, which can deliver up to 490 GB/s of bandwidth [5]. The main DDR4 memory on the same platform can deliver about 90 GB/s [5]. This memory can be configured at boot time in one of three different modes: flat, cache, or hybrid. In flat mode, the integrated memory is visible to the programmer in the same memory space as regular system memory and programmers can select which kind of memory to allocate to. In cache mode, the MCDRAM cannot be accessed directly by the programmer, but it acts as an additional level of cache in between the L2 caches and the DDR memory. The hybrid mode allows one to configure a portion of MCDRAM as flat Mode and another in cache mode each with the same characteristics just described.

4 Optimization strategies

In order to implement Eqs. 1 and 2 for supporting fully anisotropic scenarios, we have used a Finite Differences (FD) method over a Fully Staggered Grid [2] as shown in a previous work by this group [1]. Our base implementation is the direct result of developing an FD method over an FSG grid. This will lead us to a loop in time where velocities are updated based on stresses values in odd iterations and the other way around for even iterations. Velocities update involves the computation of 12 different 3D stencils plus 12 3D material interpolations for each point of the velocity grid. Materials are stored in a single vertex of the FSG cell for memory saving issues, trading storage per computation. On the other hand, stresses update consists of 28 3D stencils computation plus 84 3D interpolations for the material properties. Notice, that both velocities and stresses calculations are typically dominated by accesses to main memory to retrieve all the data needed to update the corresponding values. The following optimizations have been applied to the baseline version to improve its performance in the Intel Xeon Phi processor:

Vector Folding. This method stores a small multi-dimensional block of data in each vector size memory region compared to the single dimension in the traditional approach. Therefore memory accesses required are reduced by increasing overlap between points for the stencil computation [6].

Loop Blocking. To improve temporal data reuse and reduce the memory bandwidth requirements per updated grid point, we implemented a loop blocking strategy transforming the memory domain of our problem into smaller chunks, rather than sequentially traversing through the entire memory domain.

Software Prefetching. While the Intel Xeon Phi processor automatically prefetches data into the L1 and L2 caches it is possible to improve the memory behavior by using software prefetching. We have enabled aggressive software prefetching for L1 data cache.

Approximate Reciprocal. We approximated divisions using the reciprocal intrinsic instructions available in Intel AVX-512-ER which provide a faster implementation restraining the error.

System	Intel Xeon Phi 7250 (68 cores @ 1.4 GHz), 16 GiB of MCDRAM, 96 GiB of DDR4
Memory mode	Quadrant Flat
Compiler	Intel®C Compiler 17.0.1
CXXFLAGS	-O3 -xHost -prec-div -fp-model precise -ftz -openmp -restrict
Grid Size	$308 \times 320 \times 320$

Table 1: Evaluation environment specifications

OpenMP* Scheduling. In most implementations the default loop scheduling algorithm in OpenMP is static, but it is not necessarily the best one. We evaluated the three most relevant loop scheduling strategies: static, dynamic and guided. The best performance in our case resulted from using dynamic scheduling.
Streaming Stores. These processor instructions speed up performance in the case of vector-aligned stores. The main goal is to avoid wasting memory bandwidth by being forced to read the original content of an entire cache line from memory, when we are actually replacing the whole content.
Cooperative Threading. OpenMP parallel regions can be nested inside each other. Enabling nested parallelism helped distribute the workload among threads.
Threads per core. On Intel Xeon Phi processors is possible to place up to 4 threads per core. Reducing the numbers of threads per core to just 1 gave us the best performance.

5 Results

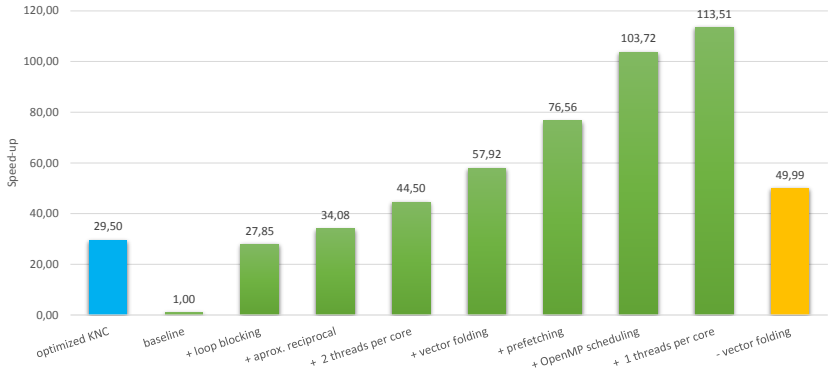

Figure 2: Elastic FSG wave propagator performance

The proposed optimizations were implemented and evaluated for a computational grid built using FSG cells. Table 1 shows the environment used for

the evaluation. Some of the proposed optimizations depend on the tuning of several settings, e.g. a combination of the number of potential loops blocking sizes for 3D scenario with any additional context, would create an intractable set of combinations that would take years to explore. Thus, we used the YASK framework[7] and its genetic algorithm (GA) auto-tuning system to find near-optimum settings. For each desired result, the GA runs for at least three separate experiments to avoid finding a local minimum prematurely. Here, we present the best results reported from multiple experiments. Figure 2 depicts the achieved performance on an Intel Xeon Phi processor for all the proposed optimizations. The optimizations were enabled one after the other; thus the figures show the accumulative improvement of all previous optimizations. For additional reference, the performance of the same algorithm but optimized for the first generation Intel Xeon Phi product (code-named Knights Corner) is presented on [1]. Cooperative threading and streaming stores optimizations did not improve performance in the best settings combination found by the GA, thus we do not show them in Figure 2. Also, it must be noted that some optimizations are interrelated. For example, the last column in Figure 2 shows that disabling vector folding from the last version results in a drop of performance much bigger than the increase that provided when it was applied in step 4. This is because optimizations applied afterwards are not useful without vector folding.

6 Conclusions

We have shown a set of optimizations, applied to a Finite Difference Numerical method solving elastic wave propagation equations on the second generation Intel Xeon Phi processor. Moreover, the proposed scheme for solving the elastic equation supports arbitrary anisotropy at a higher computational cost when compared to more traditional approaches to address this problem. The evaluated set of optimizations ranges from memory to compute optimizations. Our fully optimized code shows a speed-up of about 4x when compared with the same algorithm optimized for the previous generation processor.

7 Acknowledgements

Authors also thank Repsol for the permission to publish the present research, carried out at the Repsol-BSC Research Center. This work has received funding from the European Union's Horizon 2020 Programme (2014-2020) and from the Brazilian Ministry of Science, Technology and Innovation through Rede Nacional de Pesquisa (RNP) under the HPC4E Project (www.hpc4e.eu), grant agreement n.° 689772.

References

[1] Diego Caballero, Albert Farres, Alejandro Duran, Mauricio Hanzich, Santiago Fernández, and Xavier Martorell. Optimizing fully anisotropic elastic propagation on intel xeon phi coprocessors. In *Second EAGE Workshop on High Performance Computing for Upstream*, 2015.

[2] Sofia Davydycheva, Vladimir Druskin, and Tarek Habashy. An efficient finite-difference scheme for electromagnetic logging in 3D anisotropic inhomogeneous media. *Geophysics*, 68(5):1525–1536, 2003.

[3] Josep de la Puente, Miguel Ferrer, Mauricio Hanzich, José E Castillo, and José M Cela. Mimetic seismic wave modeling including topography on deformed staggered grids. *Geophysics*, 79(3):T125–T141, 2014.

[4] Aki Keiiti and Paul G. Richards. *Quantitative Seismology*. University Science Books, 2003.

[5] Karthik Raman. Optimizing memory bandwidth in knights landing on stream triad. `https://software.intel.com/en-us/articles/optimizing-memory-bandwidth-in-knights-landing-on-stream-triad`, 2016.

[6] C. Yount. Vector folding: Improving stencil performance via multi-dimensional simd-vector representation. In *2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS)*, pages 865–870, Aug 2015.

[7] Charles Yount, Josh Tobin, Alexander Breuer, and Alejandro Duran. Yask–yet another stencil kernel: a framework for hpc stencil code-generation and tuning. In *Proceedings of the 6th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing held as part of ACM/IEEE Supercomputing 2016 (SC16)*, WOLFHPC'16, (in press) 2016.